Journal of Physics: Conference Series **664** (2015) 062032 doi:10.1088/1742-6596/664/6/062032

THttpServer class in ROOT

Joern Adamczewski-Musch[1] and Sergey Linev[1]

[1]GSI, RB-EE, Darmstadt, Germany

E-mail: s.linev@gsi.de

Abstract. The new *THttpServer* class in ROOT implements HTTP server for arbitrary ROOT applications. It is based on *Civetweb* embeddable HTTP server and provides direct access to all objects registered for the server. Objects data could be provided in different formats: binary, XML, GIF/PNG, and JSON. A generic user interface for *THttpServer* has been implemented with HTML/JavaScript based on JavaScript ROOT development. With any modern web browser one could list, display, and monitor objects available on the server. *THttpServer* is used in Go4 framework to provide HTTP interface to the online analysis.

1. Introduction

In many experiments online tools are required to control and monitor all stages of data taking and online analysis. Usually many different software components are involved in such set up. Each of them provides own tools and methods, which is often hard to integrate with each other.
Many online monitoring tasks can be solved with web technologies: one could use HTTP protocols for data exchange; different methods for user authentication and access control; HTML and JavaScript for interactive graphics in web browsers.

2. HTTP server for ROOT applications

An HTTP server running in the ROOT application provides direct access to its data objects without the need of any intermediate files. Any ROOT object can be streamed no sooner than an HTTP request arrives for it, and can then immediately be delivered to the browser. In the following sections the key components of the newly developed web server in ROOT are treated in detail.

2.1. Sniffer of ROOT objects

Implementing a web server for ROOT objects requires an API with a unified interface to different ROOT structures. This was realized as *TRootSniffer* class. It offers methods to browse and access (‘sniff’) objects in folders, files, trees and different ROOT collections. Any object (or object element) can be identified by a path string that can be used in an HTTP request to uniquely address the object.
By default *TRootSniffer* could access all objects reachable via *gROOT* pointer: opened files, trees, canvases, and histograms. If necessary, a user could explicitly register any object in the folders structure. *TRootSniffer* provides access not only to objects, but also to all class members by means of ROOT dictionaries.

2.2. JSON representation of ROOT objects

In ROOT a binary "streamed" representation is used to store objects in files. One could try to decode such information in web browsers with JavaScript, but this does not always work, especially in case of custom streamers. The new *TBufferJSON* class solves this problem, performing all necessary I/O operations directly on the application side. It converts ROOT objects into JSON (JavaScript Object Notation) format, which can be parsed with standard JavaScript methods. An example of the JSON representation for a *TNamed* object is shown in figure 1.

```
{
  "_typename" : "TNamed",
  "fUniqueID" : 0,
  "fBits" : 50331648,
  "fName" : "name",
  "fTitle" : "title"
}
```

Figure 1. JSON representation for *TNamed* object.

TBufferJSON class performs a special treatment for *TArray* and *TCollection* classes to provide a representation that is closer to corresponding JavaScript classes without ROOT-specific overhead. User classes with custom streamers can be equipped with special function calls, providing a meaningful representation for objects data in JSON format.

TBufferJSON can stream not only whole objects, but also specified class members. This provides access to any member of an object in a text human-readable form.

Introducing *TBufferJSON* class let perform ROOT-specific I/O code completely on the server side, delivering to JavaScript-clients ready-to-use objects.

2.3. Civetweb-based server

Civetweb [1] embeddable web server was used to implement HTTP protocol in ROOT. *Civetweb* has very compact and portable code and provides necessary functionality like multithreaded HTTP requests processing, user authentication, secured HTTPS protocol support, and so on.

The *THttpServer* class in ROOT is a gateway between HTTP engine (implemented in *TCivetweb* class) and the *TRootSniffer* functionality. *THttpServer* class takes care about threads safety: any access to ROOT objects is performed from the main thread only, preventing conflicts with application code. Access to the HTTP server can be restricted using digest access authentication method [2], supported by most browsers.

2.4. FastCGI support

FastCGI [3] is a protocol for interfacing interactive programs with web servers like *Apache*, *lighttpd*, *Microsoft ISS* and many others. Contrary to widely used *CGI* (Common Gateway Interface), *FastCGI* provides a way to handle many HTTP requests in a persistently running application. From technical point of view, *FastCGI* creates a TCP server socket used by the web server to deliver HTTP requests and receive response from a local application.

The new *TFastCgi* class of ROOT implements *FastCGI* protocol as another HTTP engine for *THttpServer* class. In fact, many HTTP engines can run with *THttpServer* simultaneously, allowing access to the same data via different protocols. Usage of *FastCGI* protocol allows integration of arbitrary ROOT application into an existing web infrastructure.

3. Accessing application data with HTTP protocol

With *THttpServer* one could use HTTP requests to directly access registered ROOT objects and their data members from any kind of shell scripts. The URL syntax of HTTP requests is used to code objects name and to provide additional arguments. For instance, *TCanvas* "c1" created in the

application will get address *http://hostname:port/Canvases/c1/* in the HTTP server. In the following sections different supported requests are described.

3.1. ***root.bin*** *request*

This request returns binary data, produced with *TBufferFile* class:

http://hostname:port/Canvases/c1/root.bin

Such representation is used when objects are stored in the ROOT binary file.
To reconstruct a ROOT object from binary representation in a C++ program one should use a function shown in figure 2:

```
TObject* Reconstruct(TClass* cl, void* webbuf, int webbufsize)
{
  TObject* obj = (TObject*) obj_cl->New();
  if (obj==0) return 0;
  TBufferFile buf(TBuffer::kRead, webbufsize, webbuf, kFALSE);
  buf.MapObject(obj, obj_cl);
  obj->Streamer(buf);
  return obj;
}
```

Figure 2. Function to reconstruct object from ***root.bin*** request. Here the buffer *webbuf* contains the data retrieved by the HTTP request. The pointer on reconstructed object is returned.

3.2. ***root.json*** *request*

This request returns an object representation in JSON format, produced by *TBufferJSON* class. To request canvas "c1" from mentioned example, one should use following syntax:

http://hostname:port/Canvases/c1/root.json

Such request can be applied not only for the object itself, but also for object members. For instance, one could request *"fTitle"* member of the canvas with request:

http://hostname:port/Canvases/c1/fTitle/root.json

For the ***root.json*** request one could provide a *"compact"* URL parameter, which defines different level of compactness:

0 – no compression, nice human-readable format (default)
1 – leading spaces are removed
2 – spaces after commas and semicolons separators are removed
3 – no new lines add (maximal compression)

For example, minimal possible size of the object output will be achieved with request:

http://hostname:port/Canvases/c1/root.json?compact=3

The *"compact"* parameter does not affect the JSON format itself – the result of JavaScript-parsing will be the same in all cases.

3.3. ***h.json*** *request*

The *h.json* request returns a hierarchical description of objects (and their properties), registered to the server. It is used in the web interface to provide a tree-like display of available objects.

3.4. ***exe.json*** *request*

This request allows execution of object methods and returns result in JSON format. Like request:

http://hostname:port/Canvases/c1/exe.json?method=GetTitle

that will return the canvas title. To enable methods execution, one should disable the default read-only mode of the server, calling *THttpServer::SetReadOnly(kFALSE)* method.

3.5. ***root.png****,* ***root.jpeg*** *and* ***root.gif*** *requests*

These requests can be applied to all objects implementing a *TObject::Draw()* method (like histograms or graphs). If requested, *THttpServer* will create a temporary canvas, draw the object on the canvas and produce an image out of it. Following parameters can be applied for request:

h – image height (in pixels)
w – image width (in pixels)
opt – draw option

For instance, to produce an image with "lego" plot of two-dimensional histogram, one could apply following request:

http:// hostname:port/Files/hsimple.root/hpx/root.png?w=700&h=500&opt=lego1

3.6. ***root.xml*** *request*

This request provides an XML representation of ROOT object, produced with *TBufferXML* class.

3.7. ***GZip compression of the requests***

For each request GZip compression of returned data can be applied. This can be achieved by simple adding of '.gz' extension to the request like:

http://hostname:port/Canvases/c1/root.json.gz?compact=3

4. User interface

HTML and JavaScript are the natural choice of user-interface implementation for web servers.

4.1. JavaScript ROOT

JavaScript ROOT [4] implements ROOT-like graphics in web browsers. It is the base of graphical user interface for the *THttpServer*. A screenshot of web browser with objects available from ROOT *tutorials/http/httpserver.C* is shown on figure 3.

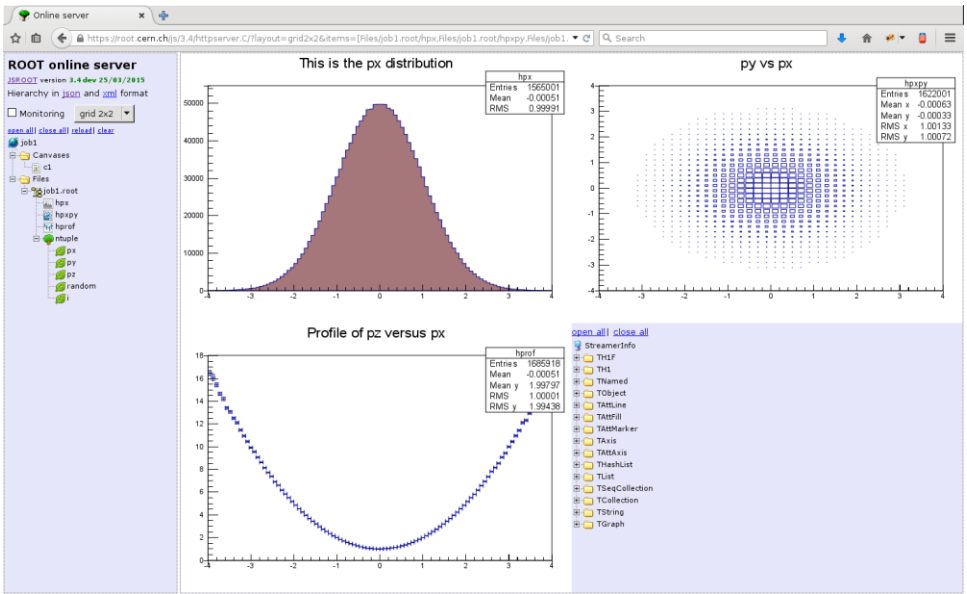

Figure 3. Browser with objects available from *tutorials/http/httpserver.C* macro. The objects hierarchy is on the left side, and several displayed histograms are on the right.

4.2. Commands execution

By default *THttpServer* provides non-destructive methods for inspection and monitoring of objects available at the server. This means that the user cannot change objects state or control execution of the ROOT code. However a command interface allows execute methods of registered objects or just perform *TROOT::ProcessLine()* on the server. ROOT application must register allowed commands to the server, which then could be triggered by user from the browser. Figure 4 contains a screenshot of the web browser with objects and commands available from ROOT *tutorials/http/httpcontrol.C* macro. The example shows how content of histograms can be reset remotely (*ResetHPX* and *ResetHPXPY* commands). Also two other commands (*Start* and *Stop*, shown only as buttons) provide control over main loop execution.

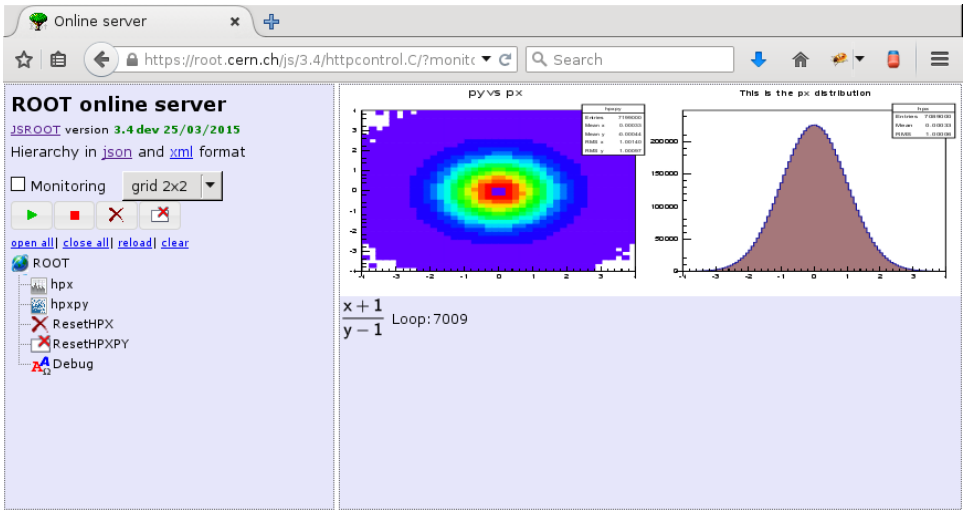

Figure 4. Browser with objects available from *tutorials/http/httpcontrol.C* macro. Registered commands (ResetHPX, ResetHPXPY) and also shortcut buttons for them are displayed in left panel.

4.3. Remote TTree::Draw

JavaScript ROOT supports reading of different ROOT classes, but it was never aimed to implement full ROOT functionality in JavaScript. Especially this is true for the *TTree* class, the most complex component of the ROOT framework. *THttpServer* provides a possibility to use *TTree::Draw()* functionality remotely from the web browsers. A special GUI element is provided which is activated when a *TTree* element is clicked in the web browser as shown on figure 5. One could specify several parameters (like draw expression or number of entries to process), used then with appropriate ***exe.json*** request. The resulting histogram is drawn then with normal JSROOT methods.

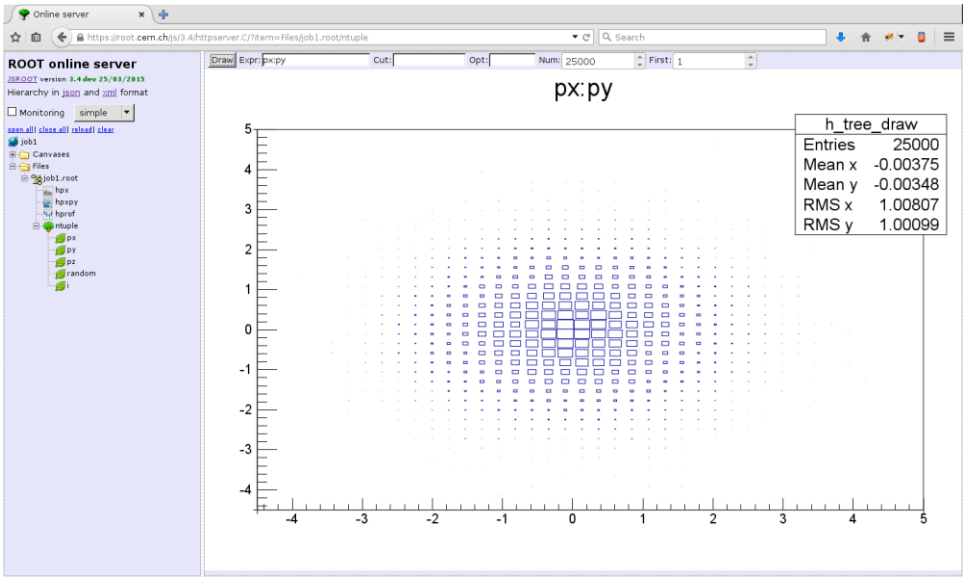

Figure 5. Example of TTree::Draw usage with *tutorials/http/httpserver.C* macro. On top an input fields for function parameters, below the result histogram.

5. THttpServer in Go4 framework

Go4 [5] is GSI software framework, based on ROOT and Qt. It includes analysis framework closely coupled with MBS [6] and DABC [7] data acquisition systems. Go4 provides powerful Qt-based GUI to monitor and steer analysis execution.

5.1. Go4 analysis control via HTTP server

A special *TGo4Sniffer* class (subclass of *TRootSniffer*) has been implemented. It registers all objects used in Go4 analysis to the *THttpServer*. These are histograms, conditions, parameters, input and output events. *TGo4Sniffer* also offers several commands for control of the Go4 analysis via HTTP protocol. For custom Go4 classes (like *TGo4Condition* or *TGo4Parameter*) JavaScript-based draw functions are provided allowing viewing and modification of these objects in the web browsers. An example of a web browser with several Go4 elements is shown in the figure 6.

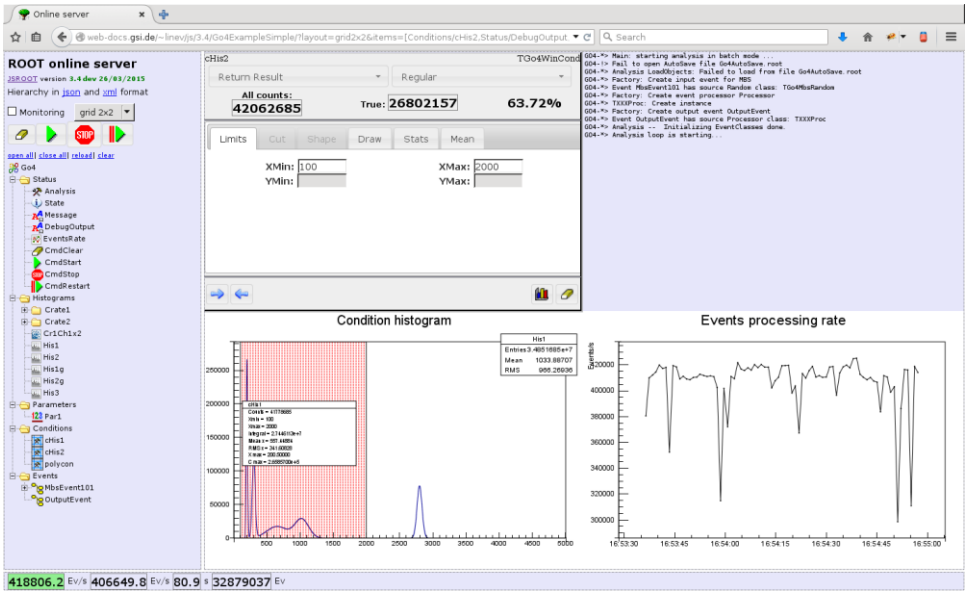

Figure 6. Browser with objects available from *Go4ExampleSimple* analysis. *TGo4Condition* editor is shown in top left panel, graphical view of condition on bottom left panel. Analysis log output is shown on the top right, event processing rate on the bottom right. Go4 commands (start/stop/restart analysis, clear objects) are provided as shortcut buttons on the of the objects browser. A status line at the bottom displays current analysis event statistics.

5.2. Go4 GUI as browser for THttpServer

HTTP protocol makes it possible to access data from *THttpServer* by different applications – not only with web browsers. Go4 GUI has been equipped with a component to read and draw objects from an arbitrary *THttpServer* instance (shown on figure 7). To transport object data between server and the Go4 GUI a binary representation is used (produced with ***root.bin*** request); decoding of objects data is performed with conventional ROOT methods. For objects display regular ROOT graphics is used. This is an advantage for complex applications where web browser graphics does not exist or does not provide enough performance.

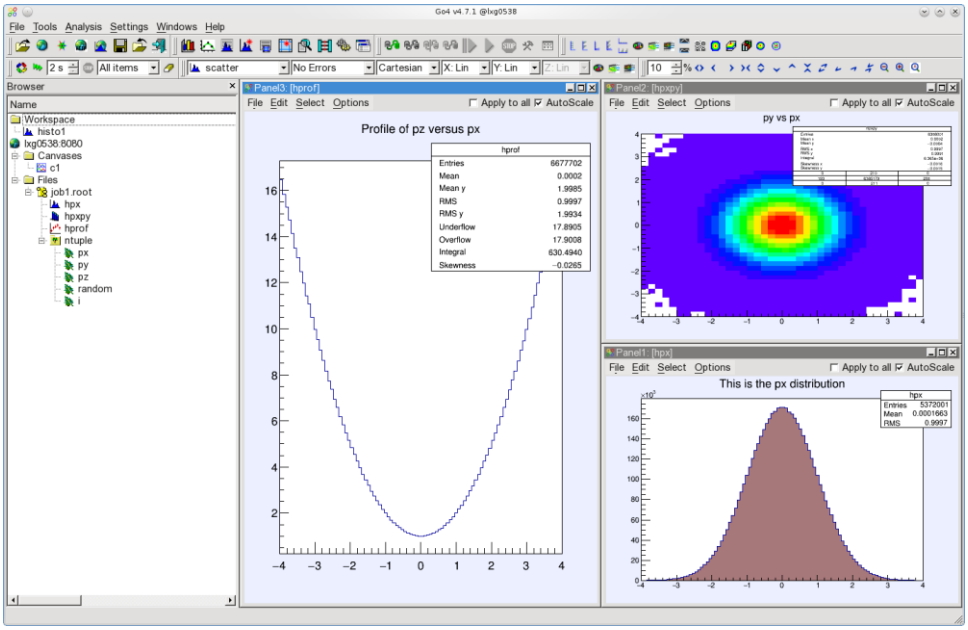

Figure 7. Qt-based Go4 GUI with objects available from ROOT *tutorials/http/httpserver.C* macro. Objects hierarchy is on the left side, several displayed histograms on the right.

6. Conclusion

THttpServer class provides HTTP access to arbitrary ROOT-based application. JavaScript and HTML code for browsing and display of different object kinds is implemented. With minimal efforts any existing ROOT application can be equipped with an HTTP server and monitored from web browser.
The code is available in both *master* and *5-34-00-patches* branches of ROOT [8] and provided with latest ROOT releases.

7. References

[1] Civetweb homepage and repository, https://github.com/bel2125/civetweb
[2] RFC 2617, HTTP Authentication: Basic and Digest Access Authentication, http://tools.ietf.org/html/rfc2617
[3] FastCGI homepage, http://fastcgi.com
[4] JavaScript ROOT homepage, https://root.cern.ch/js/
[5] Go4 homepage, http://go4.gsi.de
[6] MBS homepage, http://daq.gsi.de
[7] DABC homepage, http://dabc.gsi.de
[8] ROOT git repository, https://root.cern.ch/gitweb?p=root.git